WYŻSZA SZKOŁA BANKOWA
w Poznaniu

DEPARTMENT OF COMPUTER SCIENCE

The Mango Messenger

DIPLOMA PROJECT

Poznan, 2022

Contents

1 Project Assumptions **2**
1.1 Project description . . . 2
1.2 Project objectives . . . 2

2 Implementation **4**
2.1 Project tasks . . . 4
2.2 Project implementation . . . 6
2.2.1 Theoretical assumptions . . . 6
2.2.2 Description of facts . . . 8
2.2.3 Empirical research . . . 9
2.2.4 System Requirements . . . 13
2.2.5 Web service architecture . . . 14
2.2.6 Authorization mechanism . . . 20
2.2.7 End-to-end encryption . . . 26
2.3 Project outcomes . . . 32
2.4 Usefulness of project . . . 37
2.5 Project self-evaluation . . . 38

Bibliography **40**

WYŻSZA SZKOŁA BANKOWA
w Poznaniu

Details of the partners

A1. Mentor's details

First name and surname	Szymon Murawski
Degree	Dr.
Date and signature	

A2. Team members' details

First name and surname	Petro Kolosov
Course of study	Computer Science
Type of study program	Daytime
Date and signature	

First name and surname	Serhii Holishevskyi
Course of study	Computer Science
Type of study program	Daytime
Date and signature	

First name and surname	Illia Zubachov
Course of study	Computer Science
Type of study program	Daytime
Date and signature	

First name and surname	Arslanbek Temirbekov
Course of study	Computer Science
Type of study program	Daytime
Date and signature	

Chapter 1

Project Assumptions

1.1 Project description

Instant messaging systems (IMS) achieve a great success and became the main mean of communication between people via an internet. Thanks to the simplicity and quickness of the message exchanging more and more people over the world start to use instant messengers on daily basis. However, such a great attention forces us to discuss another aspect of these systems, an aspect of the information security and user privacy. The high attention and wide usage of the instant messaging systems in both, commercial and non-commercial ways to be a justification for selecting the subject. The subject of current thesis is entire communication structure of IMS including cryptography, protocols, data storage, means of communications. As an object of research we consider the entire entity defined as instant messaging system, in context of modern world. Mainly, the research is done using qualitative data gathered from various sources, which listed in the references. We consider qualitative research as most suitable since that problem of security in IMS is quite classic and widely discussed in scientific community. Finally, we design and implement an instant messaging system that copes with the required functionalities and satisfies the defined security requirements, considering previous research.

1.2 Project objectives

1. To analyze the security and user privacy vulnerabilities of the instant messaging system and propose mitigations.

2. To provide the system requirements for instant messaging system, both functional and non-functional.

3. To propose web service (API's) architecture that fits the requirements.

4. To discuss an authorization mechanism that fits the requirements.
5. To discuss E2E Encryption and apply to the system.
6. To implement web service (API).
7. To implement web client.
8. To implement mobile client.
9. To implement desktop client.

Chapter 2

Implementation

2.1 Project tasks

Task 1

Task name	To implement REST web–service
Teammates involved	Petro Kolosov, Serhii Holishevskii
Entities involved to solve the task	ASP NET Core, C#, SQL, SignalR, PostgreSQL, Entity Framework, ASP NET Core Identity, JWT Libraries, NUnit, Moq, FluentAssertions, MediatR, AutoMapper, Docker, Github Actions, Azure
Short description of the task (max. 5 sentences)	To implement web–service that allows to exchange an information between the parties via an internet. This task implies the database design, realtime communication, deployments and QA.
Task completion outcomes (preparation / decisions / technical dossier, etc.)	Complete. Web–service is deployed to the Azure cloud.
Star date of task execution	23-Jun-2021
End date of task execution	8-Dec-2021

Task 2

Task name	To implement web–client application
Teammates involved	Petro Kolosov, Serhii Holishevskii, Arslanbek Temirbekov, Illia Zubachov
Entities involved to solve the task	Angular 11, TypeScript, Docker, Github Actions, Azure
Short description of the task (max. 5 sentences)	Web application with specific UI that consumes REST web–service and allows exchanging various data between parties.
Task completion outcomes (preparation / decisions / technical dossier, etc.)	Complete. Web is client deployed to the Azure cloud.
Star date of task execution	5-Jul-2021
End date of task execution	10-Dec-2021

Task 3

Task name	To implement mobile client application
Teammates involved	Serhii Holishevskii, Arslanbek Temirbekov
Entities involved to solve the task	WebView, Kotlin, Android Studio
Short description of the task (max. 5 sentences)	Android application with specific UI that allows to exchange various data between parties.
Task completion outcomes (preparation / decisions / technical dossier, etc.)	Not implemented. Web–client's layout is not adaptive.
Star date of task execution	14-Dec-2021
End date of task execution	30-Dec-2021

Task 4

Task name	To implement desktop client application
Teammates involved	Petro Kolosov, Illia Zubachov
Entities involved to solve the task	ElectronJS, Javascript
Short description of the task (max. 5 sentences)	Desktop native application with specific UI that allows to exchange various data between parties.
Task completion outcomes (preparation / decisions / technical dossier, etc.)	Complete. Implemented web–client has been successfully converted to the native desktop application using ElectronJS library.
Star date of task execution	18-Jul-2021
End date of task execution	18-Jul-2021

Task 5

Task name	To implement Diffie–Hellman key exchange protocol CLI
Teammates involved	Petro Kolosov, Arslanbek Temirbekov
Entities involved to solve the task	Diffie–Hellman key exchange, Elliptic curve Diffie–Hellman key exchange
Short description of the task (max. 5 sentences)	Implementation of the command line interface which allows asymmetrically share common secret between two users.
Task completion outcomes (preparation / decisions / technical dossier, etc.)	Complete. Diffie–Hellman key exchange CLI has been implemented via REST.
Star date of task execution	15-Nov-2021
End date of task execution	21-Nov-2021

2.2 Project implementation

2.2.1 Theoretical assumptions

In the aeon of an internet instant messengers are leading instruments of communication between people in business, commercials, personal and many other aspects. However, it is very important to overview the phenomenon of an instant messaging systems from every side of it. Modern comfortable and quick means of communication may cost a

lot for unattended business as well as may act negatively to naive person using IMS first time. Recent data leaks of 533 million Facebook users [Holmes, 2021] yet again warns us on potential privacy issues in IMS. Instant messaging system is under many of dangers, from the first glance there are: phishing, virus spread, insecure communication protocols, wrong setup of host's SSL certificate, missing end-to-end encryption etc. Although, these issues are looking quite important in terms of user privacy, the most user's danger is user himself. Bruce Schneier's famous citation

> *Amateurs hack systems, professionals hack people*

reflects an entire power of social engineering approach [Luo et al., 2011], which is quite succeeded nowadays. Therefore, it is always worth to keep an eye on the following recommendation for both, commercial and non-commercial usages of an IMS

- **Ensure password strength.** Make sure that instant messaging system account you use for has enough strong password such that meets Carnegie Mellon University [Shay et al., 2010] recommendations for strong passwords.
- **Keep updated.** Always download and install an updates from your instant messenger provider, often such updates are about security and user privacy.
- **Prefer automatic updates.** Keep on automatic updates for your instant messaging client and install updates as soon as they are released.
- **Stay encrypted.** Conduct research on encryption system of IMS. Ensure that instant messenger you are using supports end-to-end encryption.
- **Do not "remember" password.** Keep off the feature "remember my password" of your instant messaging program. Prefer to log in and logout from the system manually.
- **Check your connections.** Do not accept a messages from strangers, often these are of spam or of stealing personal data.
- **File transfers.** Prefer to share files via an e-mail, not by means of your instant messenger.
- **Do not click links.** Do not click any random links under any circumstances, even you know the sender personally. Frequently, such links may lead to an infected web resources.

- **Protect Privacy of Sensitive Data.** Do not keep any private on sensitive data on the computer instant messenger installed on. Moreover, do not discuss any sensitive or private topics via instant messenger. Therefore, someone listening on the network can read anything said in your Instant Messaging conversation.

- **Stay virus protected.** Ensure properly implemented virus protection among with firewall rules on the target machine.

Beyond that, there are commercial usage recommendations of the IMS. Researchers at [Hindocha and Chien, 2003] conclude on the following aspects of the usage of IMS in enterprise

- **Follow best security practices.** End users and corporations should employ basic security practices and products such as intrusion detection and antivirus to mitigate the risk.

- **Wager cons and props.** Corporations at the outset should assess whether instant messaging is even a business necessity.

- **Support.** Enterprise versions of the instant messaging products should be utilized and administrators should be on the lookout for future enterprise security solutions that specifically address instant messaging threats.

- **Run under VPN.** Enterprise instant message system should be served under VPN.

2.2.2 Description of facts

Analyzing the topic of the end user needs, we confirm that these are quite classic, such as quick, comfortable and secure information exchange, including not only text message but as well images, files, hyper-refs. The same needs hold for commercial approach, however with slightly stronger security requirements. In order to reach desirable quality level of the application, it is worth to familiarize with an experience of the products available on market such that address the same problematic. Fon instance, currently the applications like *Microsoft Teams, Slack, Discord, Telegram, WhatsApp* are on the competitive level and fit for commercial and non-commercial usages. For example, telegram bots may be successfully used in business [Ilchenko, 2017].

2.2.3 Empirical research

We would like to start our thesis with a general description of the communication structure between an actors among with possible security and user privacy vulnerabilities. Communication model of the instant messaging system is a quite large topic since it implies various protocols, approaches etc, therefore current discussion asserts the communication over HTTP protocol via REST API using JSON data format. Since that main task of this thesis is to implement software components that meet the specified security requirements, namely: *Web Client, Web API, Mobile client, Desktop client* which are considered to be the actors we focus our attention to. The following diagram describes the basic concept of the system and conveys the relationships between the actors mentioned above.

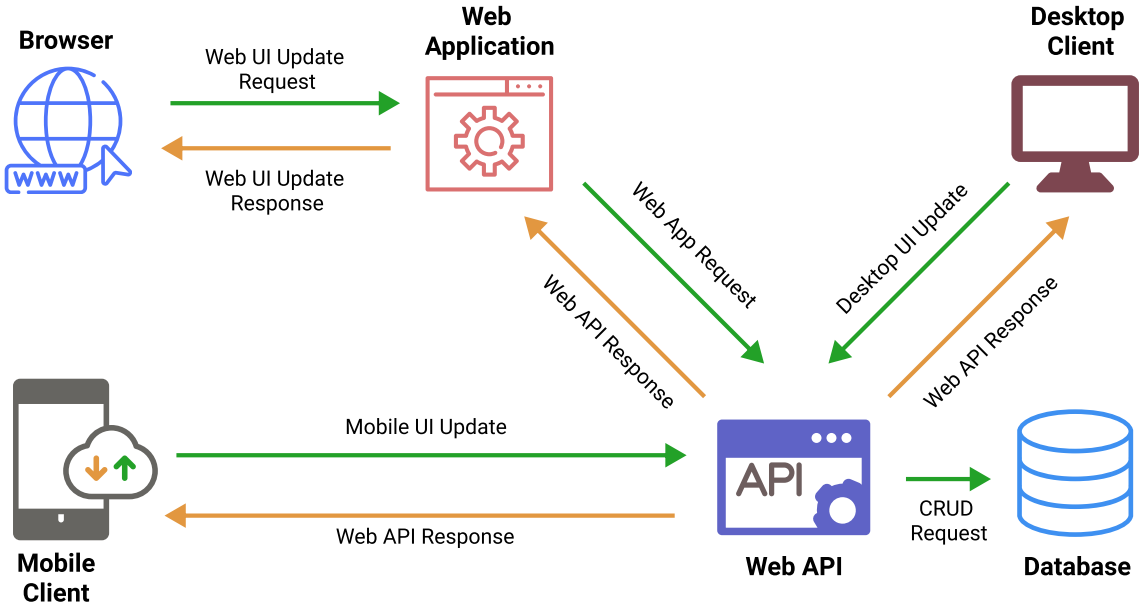

FIGURE 2.1: Software modules communication diagram. Source: [Mango Messenger Figma, 2021].

Hence, communication between software components is organised as follows

Browser – Web Application – Web API – Database communication model

- Browser downloads application static files from Web Application server.
- Browser sends a request to Web API.
- Web API checks access rights, executes business logic referring to the Database.

- Web API responds to the Browser.
- Browser user interface is being updated.

Desktop Client – Web API – Database communication model

- Desktop Client sends a request to update the user interface.
- Web API checks access rights, executes business logic referring to the Database.
- Web API responds to Desktop Client.
- Desktop Client's user interface updated as per response from the Web API.

Mobile Client – Web API – Database communication model

- Mobile Client sends a request to update the user interface.
- Web API checks access rights, executes business logic referring to the Database.
- Web API responds to the desktop Mobile Client.
- Mobile Client's user interface updated as per response from the Web API.

However, such communication models are under possible security vulnerabilities, most of which are already fixed *'out of the box'* in modern web frameworks, so we discuss these are requiring additional attention. The first vulnerability that comes to mind is phishing [Dhamija, Tygar, and Hearst, 2006]. An attacker could launch his own web application consuming our web API, therefore it is possible to log user actions and get access to personal data or credentials. Phishing attack could be mitigated using a properly configured Cross-Origin Resource Sharing [Gibbins, n.d.] policy that will restrict the queries from the domains that do not meet the policy. For instance, in our project the CORS configured as follows

```
public static void Configure(
IApplicationBuilder app,
IWebHostEnvironment env)
{
    ...

    app.UseCors(CorsPolicy);
```

```
    ...
}

public void ConfigureServices(IServiceCollection services)
{
    ...

    services.AddCors(options =>
    {
        options.AddPolicy(CorsPolicy, builder =>
        {
            var allowedOrigins = Configuration
                    .GetSection("AllowedOrigins")
                    .Get<string[]>();

            builder.WithOrigins(allowedOrigins)
                   .AllowAnyMethod()
                   .AllowCredentials()
                   .AllowAnyHeader();
        });
    });

    ...
}
```

The next potential vulnerability is improper SSL certificate configuration [Georgiev et al., 2012; El-Hajj, 2012] or usage of self-signed certificate [Kappenberger, 2012], to eliminate the vulnerability of the improper SSL certificate, it is recommended to follow the instructions and best practices [Rapp, 2021].

In addition, a potential vulnerability lies in the possibility of SQL injection [Halfond, Viegas, Orso, et al., 2006]. The SQL injection vulnerability is eliminated by using parameters in string literals of the SQL query. Also, it is necessary to pay attention to the configuration used ORM [Tiwari and Tiwari, 2015].

There is another danger that attacker may receive information about the application infrastructure through the error messages in the response from the server, thus, it is recommended to use the unified response format according to RFC 7231-Hypertext Transfer

Protocol [Fielding and Reschke, 2014]. Therefore, in case of an error response will not contain any details.

In order to provide proper authorization, it is recommended to use the roles for users in order to restrict unauthorized access to the resources available only to administrators.

The last, but not the least possible vulnerability – is a famous worm and virus spreading problem [Mannan and Van Oorschot, 2005]. Obviously, it is not a problem to get rid of the worms in local network with just a few devices connected, however worms are really dangerous for the huge networks, like messenger considered to be. The fight against the worms generally dependent on the end-user's security best practices education (at least user should not use the public unprotected wi-fi networks) and firewall settings of the network. However, the spread of viruses may be mitigated by the certain validation rules upon file upload such as follows

```
public class CommonFileValidator : AbstractValidator<IFormFile>
{
    private readonly List<string> allowedExtensions = new()
    {
        "jpg", "JPG", "txt", "TXT", "pdf",
        "PDF", "gif", "GIF", "png", "PNG"
    };

    public CommonFileValidator()
    {
        RuleFor(x => x.Length)
        .Cascade(CascadeMode.Stop)
        .GreaterThan(0)
        .LessThanOrEqualTo(5 * 1024 * 1024)
        .WithMessage("File should not exceed 5 MB.");

        RuleFor(x => x.FileName)
        .Cascade(CascadeMode.Stop)
        .NotEmpty()
        .Must(HaveAllowedExtension)
        .WithMessage("File extension is not allowed. " +
            $"Allowed extensions: {string.Join(", ", allowedExtensions)}.")
        .Length(1, 50);
```

```
    }

    private bool HaveAllowedExtension(string str)
    {
        var extension = str.Split('.').Last();
        return _allowedExtensions.Contains(extension);
    }
}
```

2.2.4 System Requirements

Prior to software module implementation, it is essentially important to define the functionality module will obtain. In this section we discuss functional and non-functional requirements of secure instant messaging system from customer's prospective. Generally, there are three forms of software product requirements: business, functional, and non-functional. Business requirements [Dilworth and Kochhar, 2007] typically answer how the product will address the needs of your company and its users. They also reveal the business model of the app and what problems it can solve. Functional requirements [Malan, Bredemeyer, et al., 2001] are about functionalities that will be implemented in the application. Non-functional requirements [Chung et al., 2012] describe how these functionalities will be implemented.

Mostly common and simple way to define software product's functional requirements are User Stories. User stories [Cohn, 2004] should be understandable both to developers and to you as the client, and should be written in simple words. The most popular way of writing a user story is with the following formula

```
"As a <user type>, I want <goal> so that <reason>."
```

Now, let's group the main features of the application as follows

- Registration
- Authentication
- Managing contacts
- Sending messages and media to individuals

- Creating and managing groups
- Sending messages and media to groups
- Viewing messages history
- Managing profile settings
- Navigation

In order not to overfill the document, an entire list of requirements, both functional and non-functional are moved to the annexes.

2.2.5 Web service architecture

Implementing the instant messenger system, we consider applying a well-known N-tier Monolithic Architecture [Bucchiarone et al., 2018], which provides a time-proven model that allows software developers to create flexible and reusable applications.

However, during the implementation of monolith it is very important to avoid the cases of crucial over-engineering of the system that leads to useless complication of the code base. For the developers, it is a vital point to follow the KISS [Alwin and Beattie, 2016] and YAGNI [Da Silva et al., 2018] software development principles in order not to reach thousands lines of code in a single class.

One would suggest to use nowadays popular Microservices Architecture, thinking about scalability [Brataas and Hughes, 2004], an ability of the system to handle large numbers of users distributed over geographically large areas without notably affecting the overall performance of the system. However, the effect of Microservices is being felt only for quite large and complex systems, not the case of our yet simple application. According to Martin Fowler [Fowler, 2015b],

> *You shouldn't start a new project with microservices, even if you're sure your application will be big enough to make it worthwhile.*

which is so-called *Monolith first* approach. Makes sense to begin an implementation from *Modular monolith*, a monolith with minimized coupling between the software components, where splitting to microservice won't be a time and financial expensive operation. Following plot demonstrates the relation between the complexity and profits between monolith and microservices

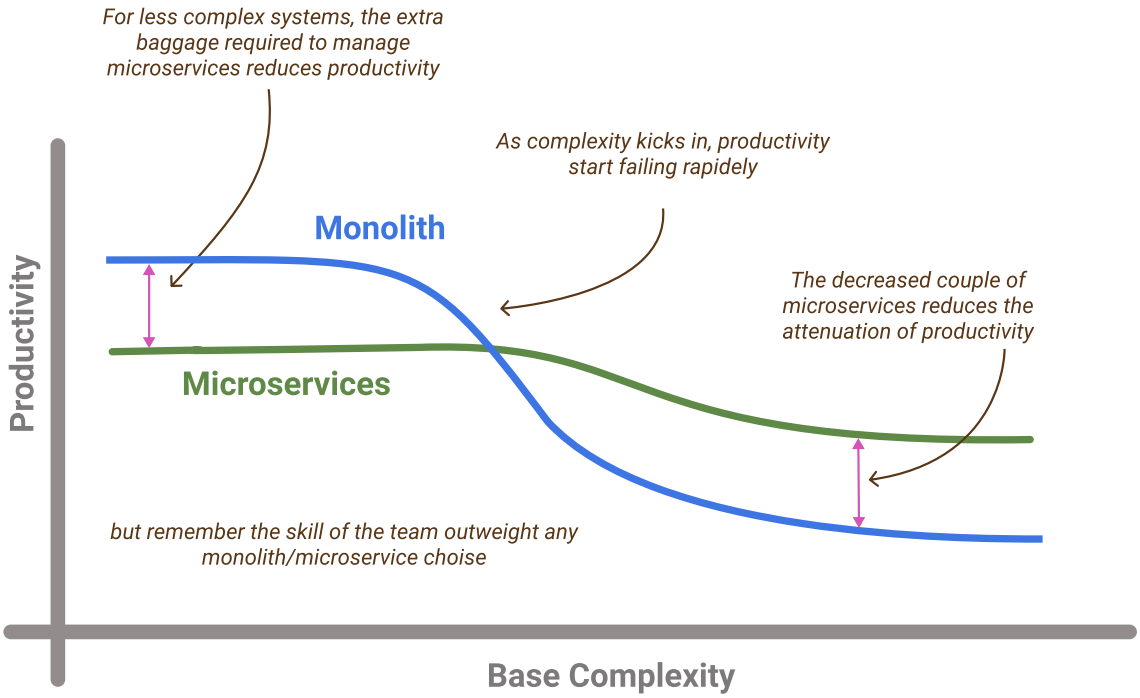
For less complex systems, the extra baggage required to manage microservices reduces productivity
As complexity kicks in, productivity start failing rapidely
Monolith
The decreased couple of microservices reduces the attenuation of productivity
Microservices
but remember the skill of the team outweight any monolith/microservice choise
Productivity
Base Complexity

FIGURE 2.2: Relation between system complexity and architectures. Source: [Fowler, 2015a].

A layered architecture usually consists of Presentation layer, Business logic layer, Data access layer. By segregating the project into layers, developers reach the options to modify or add a specific layer without reworking the entire application.

- *Presentation Layer.* Graphic user interface or API gateway.
- *Application Logic.* Encapsulates the means of interaction with user. For example, push-notifications e-mail notification, sms notifications etc.
- *Business Logic.* Encapsulates the logic of clint's request handling. For example, service layer.
- *Data Access Layer.* Responsible for logging, database access and other services required to support Business Logic layer.

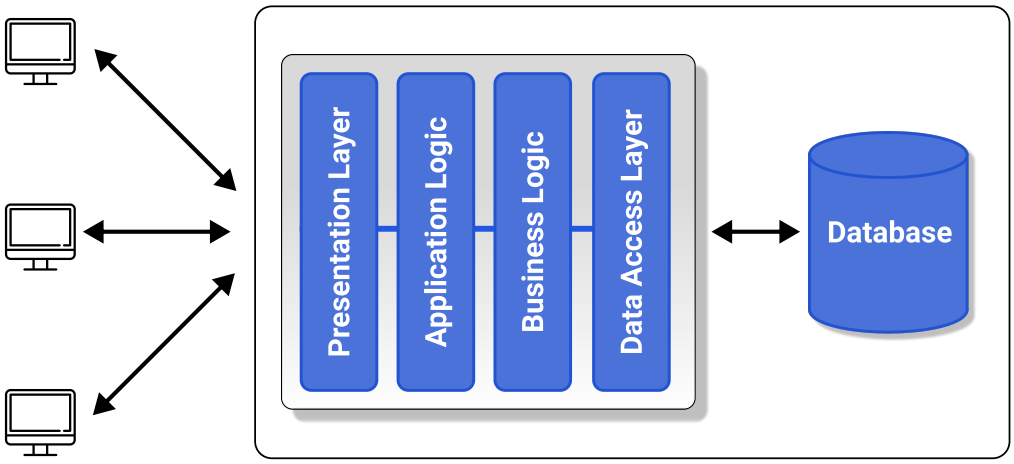

FIGURE 2.3: Monolith concept diagram. Source: [Mango Messenger Figma, 2021].

Monolithic Architecture: Cons and Props. A monolith is built as a large system with a single code base and deployed as a single unit, usually behind a load balancer. Monoliths offer several advantages, particularly when it comes to operational overhead requirements. Here are some of those basic benefits:

- *Simplicity.* Monolithic architectures are simple to build and deploy. These applications can scale horizontally, by running several copies of the application behind a load balancer. With a single codebase, monolithic apps can easily handle cross-cutting concerns, such as logging, configuration management and performance monitoring. Another advantage associated with the simplicity of monolithic applications is easier deployment. When it comes to monolithic applications, you do not have to handle many deployments but just one.
- *Performance.* Components in a monolith typically share memory which is faster than service-to-service communications using IPC [Proctor, 1999] or other mechanisms.
- *Easier debugging and testing.* In contrast to the microservices, monolithic applications are much easier to debug and test. Since that monolithic application is a single indivisible unit the process of end-to-end testing is much faster.

- *Easier development.* As long as the monolithic approach is a standard way of building applications, any engineering team has the right knowledge and capabilities to develop a monolithic application.

However, the drawback of monolithic architectures hides in their tight coupling. Over time, monolithic components and layers become tightly coupled and entangled, effecting management, scalability and continuous deployment. Another disadvantages of the monoliths include:

- *Understanding.* When a monolithic application's code base grows up, it becomes harder to understand. Obviously, huge code base of monolithic app is hard to manage therefore.
- *Reliability.* Entire application down may be caused by an error in every single component.
- *Updates.* Single and large code base causes the needs to redeploy an application on every single update.
- *Technology stack.* Technology stack of the monolithic app is limited by the technologies and providers used from the beginning of development. It makes technology stack changes to be expensive in terms of finances and time.
- *Scalability.* Application's components cannot be scaled independently, an entire application should be scaled.

Minimization of software components coupling. As we see, the monolith has its own disadvantages, like for instance: understanding the project structure, reliability concerns, technology stack limitations, scalability limitations. Obviously, some of these disadvantages cannot be mitigated because of the nature of the monolith. However, the complexity and coupling problem can be minimized applying certain approaches. Frequent violation of the single-responsibility principle of SOLID during implementing service components in business logics layer causes the over-complication of codebase over the time. The reason is that service components keep the huge number of methods in order to handle all possible CRUD requests to the database without any bounded context. Although, the SOLID rules are very powerful in solving designated code issues, it is necessary to apply them very carefully, since that most of them require high level of abstractions, which increases in size the code base and complicates the solution. Do

not overcomplicate the solution blindly following without any reason for instance *Open-Closed principle* or *Dependency inversion principle* from SOLID. These two principles require high abstraction layers for each service entity, keep in mind that you are not going to change data access layer provider every day, however, the code base becomes more complex and abstract. Schematically, the service entity is as follows

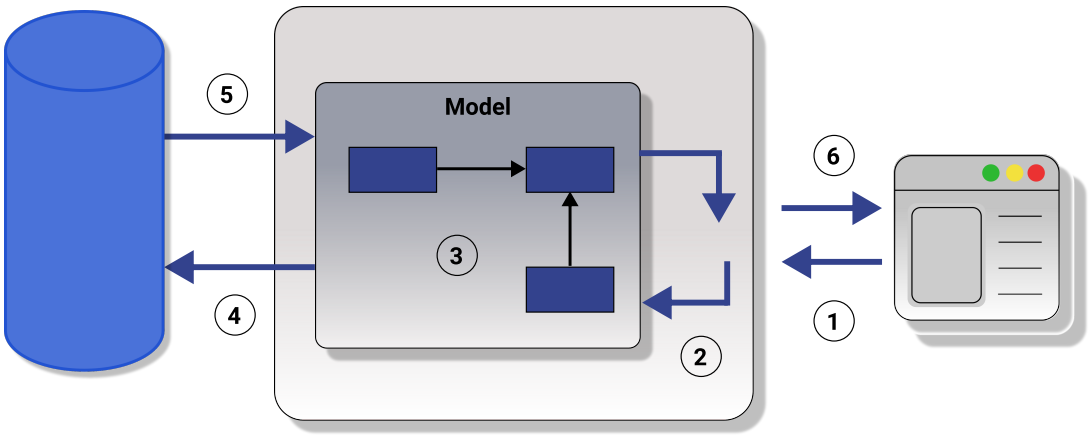

FIGURE 2.4: Service entity concept diagram. Source: [Fowler, 2011].

Where the steps are

1. User makes a change in UI.
2. Change forwarded to model.
3. Model executes validation and business logic.
4. Model updates the database.
5. Model reads from database.
6. Service updates presentation from query model.

To minimize the natural disadvantages of the monolithic architecture like complexity and high tight coupling of the components we have to recall the design patterns [Rising, 1998]. In particular, the Mediator pattern helps us to decouple the software components

from each other. Generally, Mediator is a behavioral design pattern [Rasche et al., 2016] that allows the communication between two entities, such way that entities do not know each other. Therefore, the program components depend only on a single mediator instance instead of being coupled to multiple of their colleagues. In context of the .NET platform, there are many implementations of the Mediator pattern, the most widely known and used among them is the `MediatR` library that is used in our project as well.

Another principle we are going to follow in order to minimize complexity and coupling of monolith is Command-Query Responsibility Segregation (CQRS) principle. In brief, it stands that read (query) and write (command) requests should be segregated by their responsibilities. Using CQRS and Mediator together greatly simplifies the project structure and minimizes coupling between business logic layer components. CQRS is a pattern that first described by Greg Young [Young, 2010] and its conceptual diagram as follows

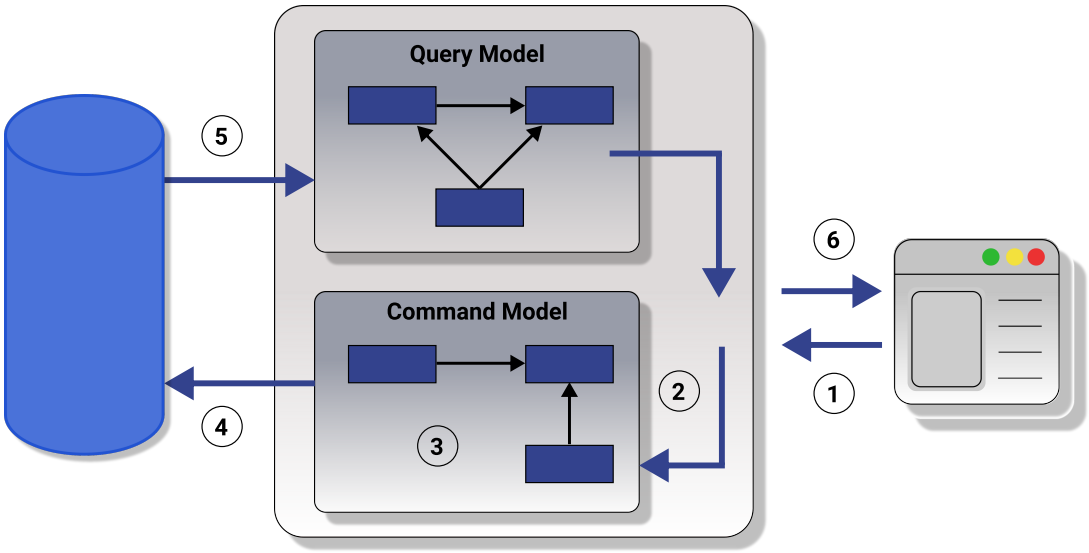

FIGURE 2.5: CQRS concept diagram. Source: [Fowler, 2011].

1. User makes a change in UI.
2. Application routes information to command model.
3. Command model executes validation and business logic.
4. Command model updates the database.
5. Query model reads from database.

6. Query service update presentation from query model.

2.2.6 Authorization mechanism

In this section the processes of Authentication and Authorization in are system discussed and explained. It is worth to remember the meaning of Authentication and Authorization definitions. Authentication – is the process of ascertaining that somebody really is who they claim to be [Burrows, Abadi, and Needham, 1989]. Authorization refers to rules that determine who is allowed to do what [Fagin, 1978]. For example, Adam may be authorized to create and delete databases, while Catherine is only authorized to read. The two concepts are completely orthogonal and independent, but both are central to security design, and the failure to get either one correct opens up the avenue to compromise. In terms of web apps, very crudely speaking, authentication is when you check login credentials to see if you recognize a user as logged in, and authorization is when you look up in your access control whether you allow the user to view, edit, delete or create content. Currently, there are two widely-known authentication methods, that are cookie authentication and JWT authentication.

JWT Tokens. JSON Web Token or JWT is an open standard [Jones, Bradley, and Sakimura, 2015] that defines a compact and self-contained way for securely exchanging information between parties as a Javascript Object Notation (JSON) object [Jones, Campbell, and Mortimore, 2015]. This information can be verified and trusted thanks to digital signature of the sender. JSON Web Tokens could be signed using a secret with the HMAC, stands for Hash-based Message Authentication Code algorithm [Wang et al., 2004] or a public-private key pair using the Rivest, Shamir and Adleman algorithm [RSA Wiener, 1990] or Elliptic Curve Digital Signature Algorithm [ECDSA Johnson, Menezes, and Vanstone, 2001]. Here are some scenarios JSON Web Tokens are useful in:

- *Authorization.* Is the most widely known scenario for using JWT. Once the user logs in, each further request will include the JWT to request header, allowing the user to access routes, services, and resources that are permitted with that token. Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains. Thanks to JWTs we are able to use Single Sign On feature, because of the JWTs' small overhead and ability to be used across different domains avoiding CORS errors.

- *Information Exchange.* JSON Web Tokens may be used as secure way of communication. Because JWTs can be signed, it is simple to verify and identify the sender.

Moreover, since that signature is calculated combining the header and the payload of the token, it is possible to verify that token has not been changed on the road.

JSON Web Token consists of the three parts separated by dots: Header, Payload, Signature. Therefore, a JWT typically looks like

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJqdGkiOiJmZDNjNjdjNS1jNmZmLTRhNWQtY
TE2Ni05OGVjZTFiNzc1MmIiLCJyb2xlIjoiVX
NlciIsIm5iZiI6MTYzMTU1MjQ5NiwiZXhwIjo
xNjMxNTUyNzk2LCJpYXQiOjE2MzE1NTI0OTYs
ImlzcyI6Imh0dHBzOi8vbWFuZ28tbWVzc2VuZ
2VyLWFwcC5oZXJva3VhcHAuY29tIiwiYXVkIj
oiaHR0cHM6Ly9tYW5nby1tZXNzZW5nZXItYXB
wLmhlcm9rdWFwcC5jb20vYXBpIn0.
locHt8ow1lFnGGZ_aFFvXI09dD4y1r594XQF2
-6YxCw
```

Let's discuss each part separately.

- *Header.* Typically, consists of two parts: the type of the token, and the signing algorithm being used, like `HMAC SHA256` or `RSA`. Example of header is as follows

```
{
    "alg": "HS256",
    "typ": "JWT"
}
```

  After that, JSON is Base64 [Josefsson, 2003] encoded to create the header part of the JWT.

- *Payload.* The second part of the token is the payload with the entity claims. Claims are statements about the user and additional data. Claims are of the following types: registered, public and private claims.

  - Registered claims. A set of predefined claims. Registered claims are not mandatory but recommended, to provide a set of useful and interoperable

claims. For example, the following are registered claims `iss` stands for issuer, `exp` stands for expiration time, `sub` stands for subject, `aud` stands for audience, and others at section 4.1 of [Jones, Bradley, and Sakimura, 2015]. The claim names are only three characters long to maintain the compactness of the JWT.

- Public claims. These claims can be defined freely. In order to avoid collisions public claims should be defined in the IANA JSON Web Token Registry or to be defined in a form of URI which contains a collision resistant namespace.
- Private claims. These claims are the claims which can be created manually in order to share the information between the parties.

An example payload could be:

```
{
    "sub": "10203040",
    "name": "Alice Fox",
    "approved": true
}
```

The payload is then Base64 encoded to form the second part of the JSON Web Token. It is recommended to put secret information in payload or header only in encrypted form, since that Base64 is encoding only and can be read by anyone.

- *Signature.* The signature part of the JWT is created combining the Base64 encoded header and payload using the specified in header algorithm. For instance, if the HMAC SHA256 algorithm is used, the signature will be created in the following way:

```
HMACSHA256(
base64UrlEncode(header) + "." +
base64UrlEncode(payload),
secret)
```

The signature is used to ensure that tokens wasn't changed during the exchange between parties, so-called Man in the middle attack.

- *Conclusions.* JWT is three Base64 encoded strings separated by dots that can be used in authorization and information exchange over HTTP, JWTs are more compact related to XML-based standards like SAML.

As to the projects concerns, we should handle multiple client applications, e.g desktop, web, mobile etc, therefore JWT authorization fits perfectly.

JWT Authorization. In authentication, when the user successfully logs in using their credentials, a JSON Web Token will be returned. Since tokens are credentials, great care must be taken to prevent security issues. In general, you should not keep tokens longer than required. You also should not store sensitive session data in browser storage due to lack of security. Whenever the user wants to access a protected route or resource, the user agent should send the JWT, typically in the Authorization header using the Bearer schema. The content of the header should look like the following:

```
Authorization: Bearer <token>
```

This can be, in certain cases, a stateless authorization mechanism. The server's protected routes will check for a valid JWT in the Authorization header, and if it's present, the user will be allowed to access protected resources. If the JWT contains the necessary data, the need to query the database for certain operations may be reduced, though this may not always be the case. If the token is sent in the Authorization header, Cross-Origin Resource Sharing (CORS) won't be an issue as it doesn't use cookies. Generally, the workflow is as follows

1. User provides credentials in order to authenticate to the system.
2. Server verifies user's authentication, fetches the login and password in database.
3. If authentication is successful, server creates session then writes this session to the database.
4. Server generates a pair of access JWT token and refresh token as GUID.
5. Server sends to client access token and refresh token.
6. Client saves the pair of access and refresh tokens.
7. User requests resource using received token passed to the request header.
8. The server check user's claims and proceeds or declines request.

The eighth point is the authorization. As a result, token stored on the client and used when it is necessary to authorize the requests. When a hacker tries to replace the data in the header or payload the token will become invalid, therefore the signature will not match the original values. So, the hacker hasn't any possibility to generate a new signature since that encryption secret key stored on the server. Access token in form of JWT is used for request authorization and for storing the additional information about user like identifier, display name and others. Refresh token in form of GUID issued by server based on successful authentication results and used to get new access-refresh token pair. Also, it is worth to add a few basic rules about JWT secure usage. The lifetime of JWT should not be long since that stolen JWT cannot be revoked. Randall Degges advices to follow the regulations [Degges, 2019]

- JWT should have a short lifetime, since it cannot be revoked.
- JWT should be used in a single time, e.g. JWT per request.

However, extremely short lifetimes of the tokens would affect the overall performance of the system. Therefore, we consider access token's lifetime to be 5 minutes and refresh token's 7 days.

For each request client preliminarily checks access token's lifetime. If access token it expired, client sends request for updating a pair of access-refresh tokens. For more confidence, we can update tokens a few seconds earlier. That is, the case when the API receives an expired access token is practically excluded. However, we are able to consider the case of interception of the request on `401UNAUTHORIZED` http status code. The following diagram demonstrates the process of requesting the resource

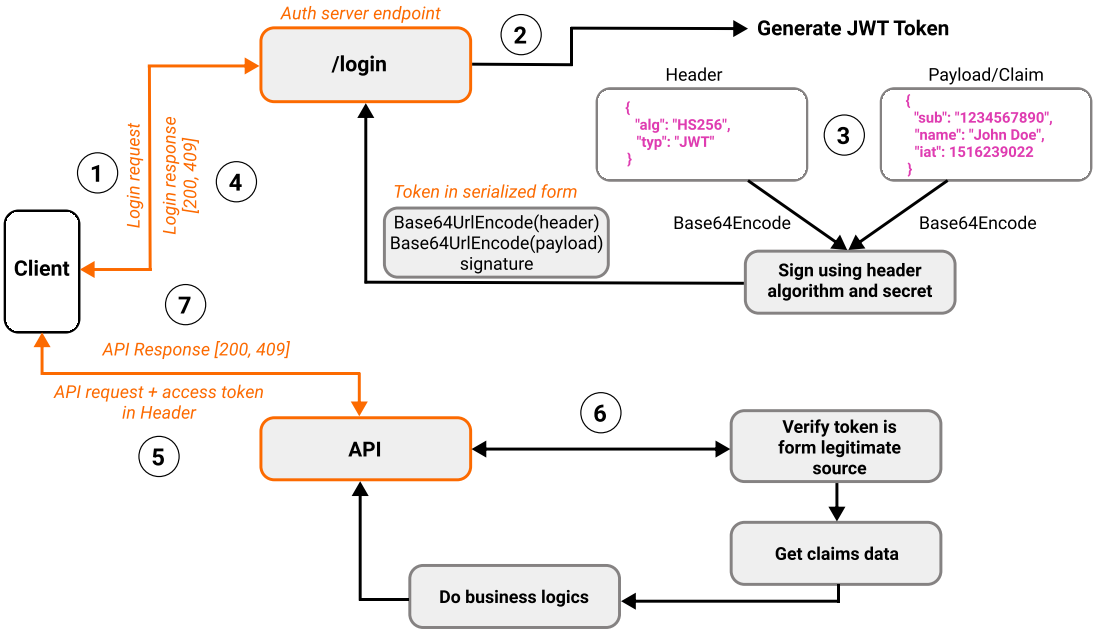

FIGURE 2.6: JWT Authorization concept diagram. Source: [Mango Messenger Figma, 2021]

By steps, the process is

- **Step 1.** *Client* application sends authentication request to the *Auth server endpoint* provided user credentials in request body.
- **Step 2.** *Auth server endpoint* responses to the *Client* with the following HTTP response codes:
  - `409CONFLICT`: Invalid credentials.
  - `200SUCCESS`: Returns a pair of access and refresh tokens.
    - **Step 3.** *Auth server* generates a pair of access and refresh tokens
      - *Auth server* fetches user data and claims.
      - *Auth server* creates new session instance in database.
      - *Auth server* Base64 encodes access token's Header.
      - *Auth server* Base64 encodes access token's Payload.
      - *Auth server* generates access token's Signature using encoded token's Header and Payload signed by means of the `HMACSHA256` algorithm and secret.

- **Step 4.** JWT access token in serialized form and refresh token in form of GUID are returned in response with `200SUCCESS` http status code to the *Client* from the *Auth server*.
- **Step 5.** *Client* queries the *API* providing access token as Bearer in request header.
- **Step 6.** *API* validates the token claims in order to authorize user
  - If authorized: *API* handles the request, goes to **Step 7**.
  - Otherwise: returns error with `401UNAUTHORIZED` http status code.
- **Step 7.** returns response with `200SUCCESS` or `409CONFLICT` http status codes to the client, according to business logic layer implementation.

2.2.7 End-to-end encryption

Historically, the process of encryption is considered to be symmetric one [Bellare et al., 1997]. It means that prior the communication, the parties conclude on the common secret key to be used in an encryption and decryption. This process is similar to the sharing keys first, then sharing the locked chest with the message. Such approach is highly cost since it requires to share the defined keys between each party taking place in secured communication. Much more simpler is to think about secured communication channel in terms of asymmetric encryption [Simmons, 1979]. The real life example would be

Alice shares with all the actors an opened lock, but keeps the key with herself.

So that Bob receives an opened lock, writes letter to Alice, puts letter to the chest, locks the chest with received from Alice lock. This way only Alice would be able to open the chest and to read the letter. This is an idea of the asymmetric encryption. However, such a simple communication concept sometimes requires complex number theory approach. A concept of opened lock may be interpreted in terms of one-way functions [Rompel, 1990]. One way function is the function that is easy to compute on every input, but hard to invert given the image of a random input. Thus, it is much simpler to close the lock without key, but very difficult to open lock combining various keys.

End-to-end encryption [Schillinger and Schindelhauer, 2019] is an asymmetric encryption such that the only communicating parties are able to decrypt the data. It means that even system administrators are not able to decrypt the messages transmitted between parties via their communication channel. End-to-end encryption can be reached via numerous approaches. Generally, there are two ways to implement E2E encryption

- Sharing public key to be used in encryption of the secret message, then encryption is done by the public key's owner, so-called asymmetric encryption. Public key owner is able then to decrypt secret message. For example, RSA algorithm.

- Asymmetric key exchange where parties exchanging the keys first, then symmetrically encrypting and decrypting the transferred data. For instance, Diffie–Hellman key exchange and AES256 encryption using common secret.

The most important aspect here is to securely store the secrets on the user's client application. Looking to the Telegram example, we can conclude that it does not make sense to implement end-to-end encryption for web and desktop clients [Job, Naresh, and Chandrasekaran, 2015; Sušánka and Kokeš, 2017; Lee et al., 2017], due to the storage security issues. Telegram uses the huge and heavy `MTProto 2.0` cryptographic protocol based on Diffie—Hellman key exchange and further AES256 symmetric encryption. According to the project concerns, the E2E encryption via Diffie–Hellman key exchange and AES256 to be considered and implemented, the next section is about.

Diffie—Hellman key exchange. Diffie–Hellman (DH) protocol is a method of asymmetric exchange of the cryptographic keys for a group of two or more participants, developed in 1976 by cryptographers Ralph Merkle, Whitfield Diffie and Martin Hellman. In contrast to symmetric key exchange, the Diffie—Hellman protocol eliminates the direct transfer of the shared secret between the participants, each participant computes a shared secret with his own private-public key pair. The Diffie—Hellman protocol is based on a one-way function of the form

$$A = G^a \bmod P \tag{2.1}$$

where A is the user's public key, a is the user's private key, $P = 2Q + 1$ is modulus, such that 2048 bits safe-prime because Q is also prime, G is generator such that G is primitive root modulo P. We say that G is primitive root modulo P if for each $1 \leq a \leq P - 1$ the $A = G^a \bmod P$ is unique and belong to the set $\{1, 2, \ldots, P - 1\}$. The period of such cyclic group $\mathbb{Z}_P$ is $P - 1$ then.

Thus, the safety of the Diffie–Hellman protocol is based on the discrete logarithm problem, which is unsolvable in polynomial time if the constants G and P are chosen correctly. Graphically, the flow of the Diffie–Hellman protocol can be expressed through the analogy with mixing paints, as below picture shows

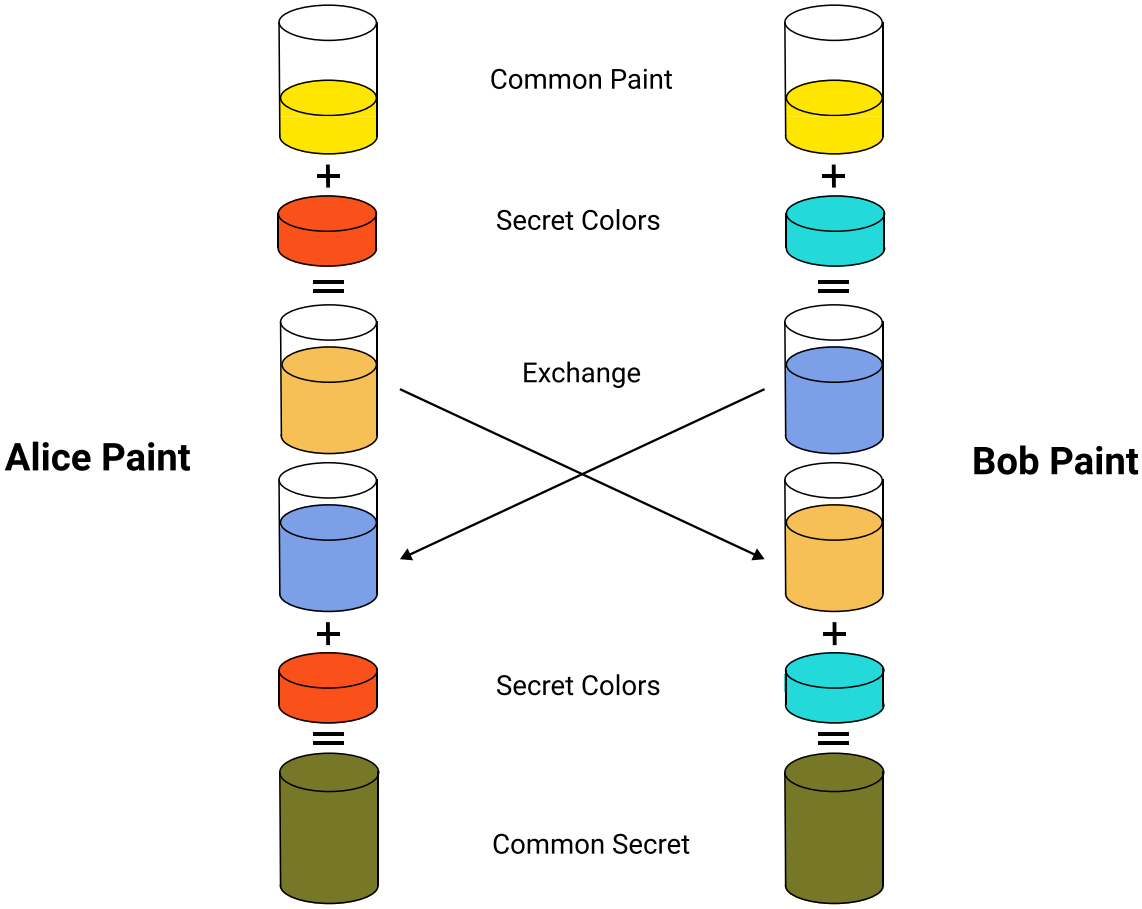

FIGURE 2.7: Diffie—Hellman key exchange concept diagram. Source: [Mango Messenger Figma, 2021].

In contrast to the Diffie—Hellman based on discrete logarithm problem, there is an Elliptic Curve Diffie—Hellman key exchange, which based on the elliptic curve discrete logarithm problem. Although, the idea is quite same, the difference only in that Elliptic Curve Diffie—Hellman ensures the same safety as discrete logarithm Diffie—Hellman with lower value of the prime modulus P. For instance, 521 bit modulus used in Elliptic Curve Diffie—Hellman is equally safe as 2048 bit modulus in discrete logarithm Diffie—Hellman. To summarize, the flow of Diffie—Hellman key exchange is as follows

1. Given 2048 bits prime modulus P and generator G, such that G is primitive root modulo P.
2. Alice chooses her secret a.
3. Alice sends to Bob her public key $A = G^a \bmod P$.

4. Bob chooses his secret b.
5. Bob sends to Alice his public key $B = G^b \bmod P$.
6. Alice computes common secret $s = B^a \bmod P$.
7. Bob computes common secret $s = A^b \bmod P$.
8. Alice and Bob have arrived to the same value

$$A^b \bmod P = G^{ab} \bmod P \tag{2.2}$$
$$B^a \bmod P = G^{ba} \bmod P \tag{2.3}$$

Diffie—Hellman key exchange implementation via REST. Although, the idea of Diffie—Hellman key exchange looks quite simple, some remarks on the concrete implementation should be added. Firstly, it is necessary to implement the mechanism of key exchange request between two or more parties. As it discussed above, each user has his own private-public keys pair, so in order to perform request between parties, it should be implemented dedicate REST [Ong et al., 2015] web–service endpoint, for instance the `POST: api/key-exchange-requests` which takes the request body of the form

```
{
    "requestedUserId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "publicKey": "RUNLMSAAAAC2lkqYcTGhutQPxcjvoqUELKoy0"
}
```

So, request sender generates on the client side a key pair, keeps private on in the file system and shares the public in request to receiver. Therefore, the second party has received the key exchange request. In order to display all the key exchange requests awaiting the confirmation of decline decisions, it is worth to implement another REST endpoint such that `GET: api/key-exchange-requests`, so that requested party will have the list of requests to proceed. This endpoint may return the data structure like follows

```
{
    "keyExchangeRequests": [
        {
            "requestId": "81d314c1-913f-4686-827e-ef2a65ccc370",
            "senderId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
            "senderPublicKey": "RUNLMSAAAAC2lkqYcTGhutQPxcjvoqUELKoy0"
```

```
        }
    ],
    "message": "SUCCESS",
    "success": true
}
```

Finally, requested party should be able to confirm or decline the key exchange request, the `DELETE: api/key-exchange-requests` endpoint should be implemented then. The server is able to fetch the request thanks to the body endpoint takes

```
{
    "requestId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "confirmed": true,
    "publicKey": "string"
}
```

Therefore, an identifier of awaiting request is passed to the server among with boolean value indicating the confirmation. Under the roof of this operation are also generation of private-public keys pair for the requested party and generation of common secret stored in client's file system. As result, the initial request sender receives a public key as confirmation from requested party. Requested side may get all his public keys via the REST web–service using the resource `GET: api/public-keys`

```
{
    "publicKeys": [
        {
            "partnerId": "ae9e10a4-0c7e-4911-8450-4139d4a114a7",
            "partnerPublicKey": "RUNLMSAAAAAbc49wfaZ+QF9J2cu1S66bkp0"
        }
    ],
    "message": "SUCCESS",
    "success": true
}
```

Now requested participant is able to derive the common secret. In order to provide an example, a simple command line interface is implemented. We have used an Elliptic Curve Diffie—Hellman implementation `ECDiffieHellmanCng Class` from the namespace `System.Security.Cryptography` of the .NET base class library. The `P-256` curve is used.

More precisely, the following CLI commands are implemented

- `MangoAPI.DiffieHellmanConsole login SENDER_EMAIL SENDER_PASSWORD`
- `MangoAPI.DiffieHellmanConsole key-exchange RECEIVER_ID`
- `MangoAPI.DiffieHellmanConsole key-exchange-requests`
- `MangoAPI.DiffieHellmanConsole confirm-key-exchange REQUEST_ID`
- `MangoAPI.DiffieHellmanConsole print-public-keys`
- `MangoAPI.DiffieHellmanConsole create-common-secret RECEIVER_ID`

Commands are self-explanatory, therefore we skip the detailed documentation on them. An example of console output straightforward

```
PS C:\Users\pkolosov> MangoAPI.DiffieHellmanConsole login $env:SENDER_EMAIL $env:SENDER_PASSWORD
Attempting to login ...
Writing tokens to file ...
Login operation success.

PS C:\Users\pkolosov> MangoAPI.DiffieHellmanConsole key-exchange $env:RECEIVER_ID
Key exchange request with an ID 5810fb94-e3ce-4d2a-a033-95366a7c2b30 created successfully.
Writing private key to file...
Writing public key to file ...
Key exchange request sent successfully.

PS C:\Users\pkolosov> MangoAPI.DiffieHellmanConsole login $env:RECEIVER_EMAIL $env:RECEIVER_PASSWORD
Attempting to login ...
Writing tokens to file ...
Login operation success.

PS C:\Users\pkolosov> MangoAPI.DiffieHellmanConsole key-exchange-requests
RequestId: 5810fb94-e3ce-4d2a-a033-95366a7c2b30
SenderId: fd3c67c5-c6ff-4a5d-a166-98ece1b7752b
Sender Public Key: RUNLMSAAAADBFEUGJcwSeMlwDi4Jf4s6IAbCvP5w0TiYB/G/iKB3IhbSgSNeG0PUje2NBdjp534psQbKf6Gup7JfM5zwxiVy

PS C:\Users\pkolosov> MangoAPI.DiffieHellmanConsole confirm-key-exchange 5810fb94-e3ce-4d2a-a033-95366a7c2b30
Writing private key to file...
Writing public key to file ...
Writing common secret to file...
Key exchange request confirmed successfully.

PS C:\Users\pkolosov> MangoAPI.DiffieHellmanConsole print-public-keys
PartnerId: fd3c67c5-c6ff-4a5d-a166-98ece1b7752b
Public Key: RUNLMSAAAADBFEUGJcwSeMlwDi4Jf4s6IAbCvP5w0TiYB/G/iKB3IhbSgSNeG0PUje2NBdjp534psQbKf6Gup7JfM5zwxiVy

PS C:\Users\pkolosov> MangoAPI.DiffieHellmanConsole login $env:SENDER_EMAIL $env:SENDER_PASSWORD
Attempting to login ...
Writing tokens to file ...
Login operation success.

PS C:\Users\pkolosov> MangoAPI.DiffieHellmanConsole create-common-secret $env:RECEIVER_ID
Writing common secret to file...
Common secret generated successfully.
```

FIGURE 2.8: Diffie—Hellman key exchange console output. Source: [Mango Messenger Figma, 2021].

Finally, both test accounts reached the same common secret.

2.3 Project outcomes

In this section project outcomes are described. Outcomes description includes the specifications of concrete functional requirements implemented along with the graphic user interface screenshots. Function requirements could be found in annexes. We begin from the messenger's start page and continue further with user contacts component and user settings component.

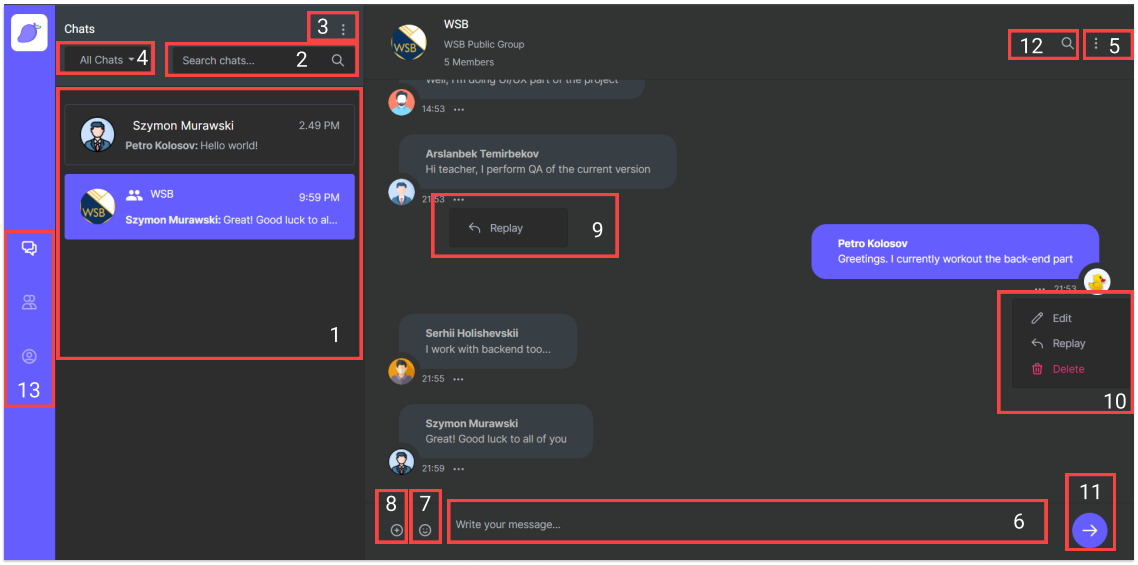

FIGURE 2.9: Start page component screenshot. Source: [Mango Messenger Figma, 2021].

1. Element responsible for getting user's chats list so that functional requirement *"As an authorized user, I want to view a message history of particular chat or group so that I see a list of my active chats on the UI"* is satisfied.

2. Element responsible for searching chats by display name and displaying it so that functional requirements

   - *"As an authorized user, I want to search public groups by title so that I enter display name to specified field, click button "Search chats" and see results"*
   - *"As a registered user, I want to join public groups so that I click button "Join group"to join the group"*

   are satisfied.

3. Element responsible for creating groups so that functional requirement *"As a registered user, I want to tap "Create channel" so that I create a new channel of the one of the types: Private channel, Public channel, Readonly channel"* is satisfied.

4. Element responsible for filtering chats so that functional requirement *"As an authorized user, I want to filter a message history of particular chat or group so that I see a filtered list of my active chats on the UI"* is satisfied.

5. Element responsible for archiving and leaving from the particular chat, so that functional requirements

    - *"As registered user, I want to tap "Leave" so that I leave from specified chat or channel"*
    - *"As a registered user, I want to tap "Archive" so that I archive the specified chat or channel"*
    - *"As a registered user, I want to tap "Un-archive" so that I un-archive the specified archived chat or channel"*

    are satisfied.

6. Element responsible for entering the text and sending a message by enter, so that functional requirement *"As an authorized user, I want to send a text message so that other members of the group see the message I sent"* is satisfied.

7. Element responsible for adding emoji to message, so that functional requirement *"As an authorized user, I want to add an emoji to the message so that other members of the group see the message with emoji I sent"* is satisfied.

8. Element responsible for adding attachments to the message, so that functional requirement *"As an authorized user, I want to add an attachment to the message so that other members of the group see the message with attachment I sent"* is satisfied.

9. Element responsible for replying to message, so that functional requirement *"As registered user, I want tap "Reply" so that I want reply to the particular message"* is satisfied.

10. Element responsible for editing and deleting a message so that functional requirements

- *"As an authorized user, I want to tap "Edit" on my message so that other members of the group see the message I edited"*
- *"As an authorized user, I want to tap "Delete" on my message so that my message is deleted for all members of the group"*

are satisfied.

11. Element responsible for sending message if message text field is not empty so that functional requirement *"As an authorized user, I want to send a text message so that another user sees my message"* is satisfied.

12. Element responsible for searching messages in the particular chat so that functional requirement *"As an authorized user, I want to search messages in particular chat so that I see the results in messages window of the chat"* is satisfied.

13. Element responsible for navigation about main page, contacts page and personal information page so that functional requirement *"As an authorized user, I want to navigate between the pages so that there is a menu on the UI"* is satisfied.

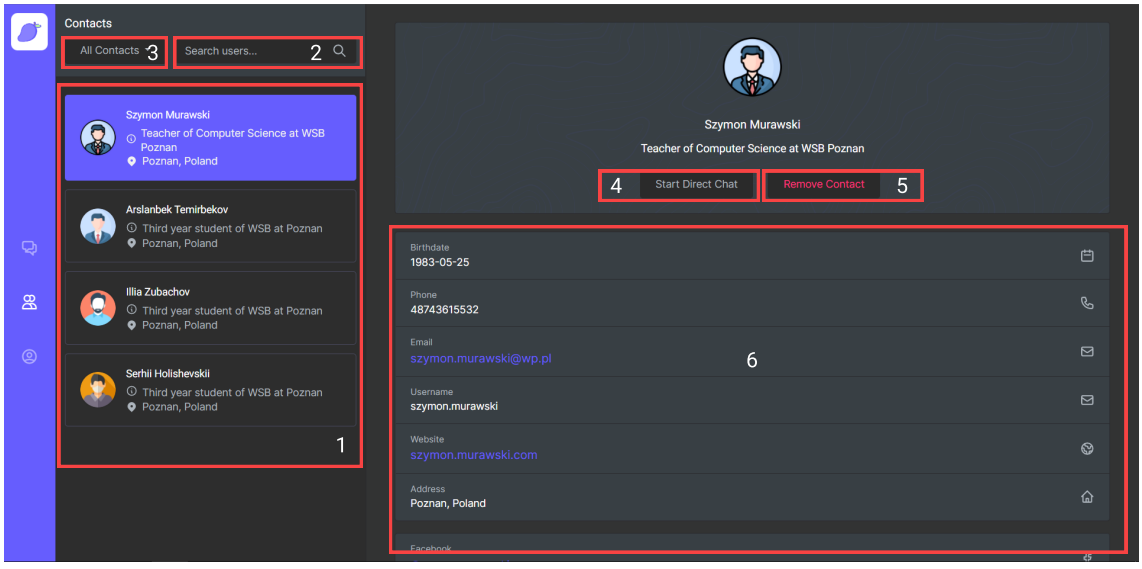

FIGURE 2.10: Manage contacts component screenshot. Source: [Mango Messenger Figma, 2021].

1. Element responsible for output user's contacts so that functional requirement *"As an authorized user, I want to see my contact list so that there is a list of users who are my contacts"* is satisfied.

2. Element responsible for searching users so that functional requirement *"As an authorized user, I want to search users so that I write user display name or phone number of e-mail address to specified input, click "Search user" button and see results"* is satisfied.

3. Element responsible for filtering contacts so that functional requirement *"As an authorized user, I want to search users so that I write user display name or phone number of e-mail address to specified input, click "Search user" button and see results"* is satisfied.

4. There is a button, clicking on which you can start chat with a specific user, so that functional requirement *"As a registered user, I want to tap "Start direct chat" so that I create a new direct chat with specified user"* is satisfied.

5. Element responsible for deleting user form contacts so that functional requirement *"As an authorized user, I want to remove the user from my contact list so that I click "Remove contact" button on user profile and remove him from my contact list"* is satisfied.

6. Element responsible for the output specified user's info so that functional requirement *"As an authorized user, I want to tap on specified contact so that I want see user's information"* is satisfied.

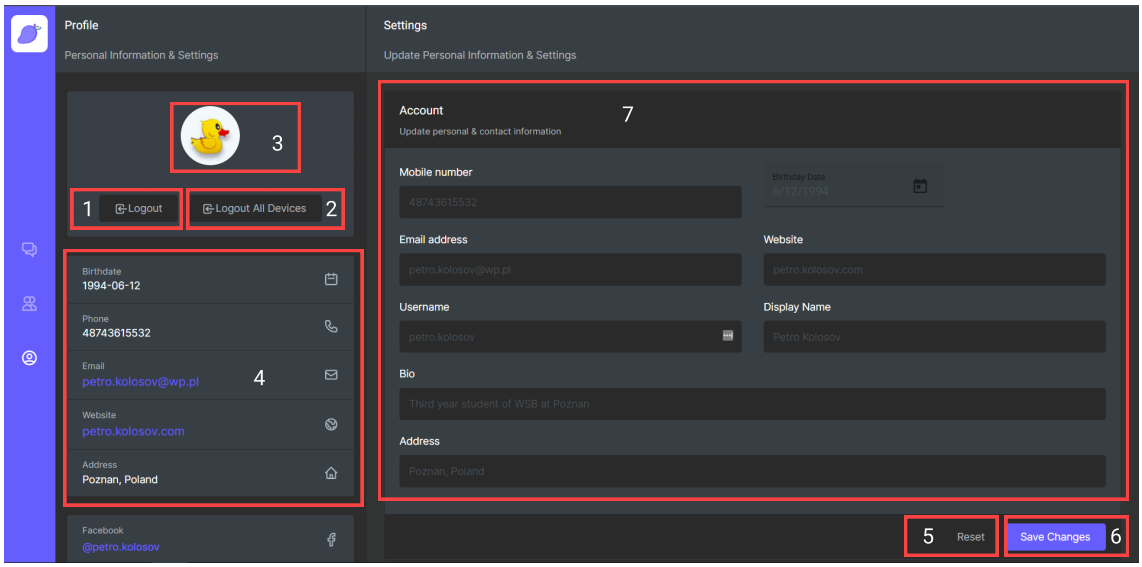

FIGURE 2.11: Account settings component screenshot. Source: [Mango Messenger Figma, 2021]

1. Element responsible for log out from specified device so that functional requirement *"As an authorized user, I want to tap "Logout" button so that current device will be logged out from the system"* is satisfied.

2. Element responsible for log out from all devices so that functional requirement *"As an authorized user, I want to tap "Logout all" button so that all my authorized devices will be logged out from the system"* is satisfied.

3. Element responsible for output user's avatar so that functional requirement *"As an authorized user, I want to navigate to personal information page so that I want see my profile picture"* is satisfied.

4. Element responsible for output user's information so that functional requirement *"As an authorized user, I want to navigate to personal information page so that I want see my personal information"* is satisfied.

5. Element responsible for reset updated personal information so that functional requirement *"As an authorized user, I want to tap "Reset" so that I want reset my updated (not saved) personal information"* is satisfied.

6. Element responsible for saving personal information so that functional requirement *"As an authorized user, I want save my updated personal information so that users see it"* is satisfied.

7. Element responsible for changing personal information so that functional requirement *"As an authorized user, I want to update my personal information in profile settings so that other users my updated personal information"* is satisfied.

Finally, we attach the list of technologies was used during an implementation of the project. List of technologies are separated by categories is as follows

- **Frameworks:** ASP .NET 5, Angular 11.2.7
- **Persistence:**
  - SQL Database: PostgreSQL 13
  - ORM: Entity Framework Core 5.0
  - Storage: Azure Blob
- **Authorization:** ASP .NET Identity Core & JWT Bearer

- **Business Logic:**
  - MediatR
  - Fluent Validation
  - AutoMapper
- **Presentation:**
  - API Documentation: OpenAPI
  - Realtime Communication: SignalR
  - Frontend Development: Angular
  - Desktop Development: ElectronJS
  - Mobile Development: WebView
- **Unit and Integration Testing:** XUnit, Moq, FluentAssertions, EntityFramework-Core InMemory
- **Code Quality Tools:** SonarQube, CodeCov
- **Containerization:** Docker
- **Continuous Integration:** GitHub Actions
- **Continuous Delivery:** GitHub Actions, Heroku, Azure
- **Programming languages:** C#, SQL, TypeScript, Kotlin
- **Tools:** Visual Studio, Rider, VS Code, WebStorm, PgAdmin, Postman

2.4 Usefulness of project

The project may be used as corporate messenger in closed scope of some company under VPN as cheaper alternative to the nowadays popular Microsoft Teams. The system provides independence from the centralized Microsoft's products. Providing a various types of subscriptions and support levels current project may be considered as corporative or educational solution.

2.5 Project self-evaluation

In this section each of the project's Authors describes his or her skills and competencies that were developed while working on the project and identifies issues encountered while working on the project.

- **Petro Kolosov.** I have obtained an experience in best practice in context of modern web applications. Got familiar with cryptographic protocols and approaches as well as encryption algorithms, both symmetric and asymmetric. Also, increased my knowledge in software modules architecture. Have worked with technologies: ASP.NET Web API, PostgreSQL 13, Entity Framework Core, ElectronJS, NUnit, Moq, SonarQube, Docker, Heroku, Azure, C#, SQL, TypeScript. Moreover, familiarized myself with CI/CD practices, precisely, writing pipelines for various environments, like Azure. Faced the following issues
  - Have been engaged in backend implementation using ASP .NET
  - Written CI/CD pipelines for Angular Front-End Application, Backend, Thesis document deployment on GitHub Pages
  - Discussed database structure
  - Written thesis document using LaTeX
  - Set up environments: QA, Dev. Where QA is deployed on Heroku and Dev environment is deployed on Azure.
- **Serhii Holishevskii.** I have obtained an experience in best practice in context of modern web applications. Got familiar with cryptographic protocols and approaches as well as encryption algorithms, both symmetric and asymmetric. Also, increased my knowledge in software modules architecture. Have worked with technologies: ASP.NET Web API, PostgreSQL 13, Entity Framework Core, ElectronJS, NUnit, Moq, SonarQube, Docker, Heroku, Azure, C#, SQL, TypeScript. Faced the following issues
  - Have been engaged in backend implementation using ASP .NET
  - Written CI/CD pipelines for Angular Front-End Application, Backend, Thesis document deployment on GitHub Pages
  - Discussed database structure
  - Written thesis document using LaTeX

  - Implemented mobile client application using WebView
  - Front-end development using Angular and TypeScript

- **Illia Zubachov.** I have gained a knowledge of modern frontend web frameworks such as Angular, Angular materials. Got experience working with Typescript programming language. Also, have familiarized myself with QA best practices and approaches. Worked with technologies: TypeScript, Angular, ElectronJS, Docker. Faced the following issues
  - Front-end development using Angular and TypeScript
  - Unit tests writing for backend
  - QA of the front end project
  - Discussed database structure
  - Written thesis document using LaTeX
- **Arslanbek Temirbekov.** I have gained a knowledge of modern frontend web frameworks such as Angular, Angular materials. Got experience working with Typescript programming language. Also, has familiarized myself with QA best practices and approaches. Worked with technologies: TypeScript, Angular, ElectronJS, Docker.
  - Front-end development using Angular and TypeScript
  - Integration tests writing for backend
  - QA of the front end project
  - Discussed database structure
  - Written thesis document using LaTeX

Bibliography

Alwin, Duane F and Brett A Beattie (2016). "The KISS principle in survey design: question length and data quality". In: *Sociological methodology* 46.1, pp. 121–152.

Bellare, Mihir et al. (1997). "A concrete security treatment of symmetric encryption". In: *Proceedings 38th Annual Symposium on Foundations of Computer Science*. IEEE, pp. 394–403.

Brataas, Gunnar and Peter Hughes (2004). "Exploring architectural scalability". In: *Proceedings of the 4th international workshop on Software and performance*, pp. 125–129.

Bucchiarone, Antonio et al. (2018). "From monolithic to microservices: An experience report from the banking domain". In: *Ieee Software* 35.3, pp. 50–55.

Burrows, Michael, Martin Abadi, and Roger Michael Needham (1989). "A logic of authentication". In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 426.1871, pp. 233–271.

Chung, Lawrence et al. (2012). *Non-functional requirements in software engineering*. Vol. 5. Springer Science & Business Media.

Cohn, Mike (2004). *User stories applied: For agile software development*. Addison-Wesley Professional.

Da Silva, Tiago Silva et al. (2018). "The evolution of agile UXD". In: *Information and Software Technology* 102, pp. 1–5.

Degges, Randall (2019). *JWTs Suck*. https://speakerdeck.com/rdegges/jwts-suck. [Online; accessed 15-August-2021].

Dhamija, Rachna, J Doug Tygar, and Marti Hearst (2006). "Why phishing works". In: *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pp. 581–590.

Dilworth, John and AK Kochhar (2007). "Creation of an e-business requirements specification model". In: *Journal of Manufacturing Technology Management*.

El-Hajj, Wassim (2012). "The most recent SSL security attacks: origins, implementation, evaluation, and suggested countermeasures". In: *Security and Communication Networks* 5.1, pp. 113–124.

Bibliography

Fagin, Ronald (1978). "On an authorization mechanism". In: *ACM Transactions on Database Systems (TODS)* 3.3, pp. 310–319.

Fielding, Roy and Julian Reschke (2014). "RFC 7231-Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content". In: *Internet Engineering Task Force (IETF)*.

Fowler, Martin (2011). *CQRS*. https://martinfowler.com/bliki/CQRS.html. [Online; accessed 20-December-2021].

— (2015a). *MicroservicePremium*. https://martinfowler.com/bliki/MicroservicePremium.html. [Online; accessed 20-December-2021].

— (2015b). *MonolithFirst*. https://martinfowler.com/bliki/MonolithFirst.html. [Online; accessed 20-December-2021].

Georgiev, Martin et al. (2012). "The most dangerous code in the world: validating SSL certificates in non-browser software". In: *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 38–49.

Gibbins, Nicholas (n.d.). "Cross Origin Resource Sharing". In: ().

Halfond, William G, Jeremy Viegas, Alessandro Orso, et al. (2006). "A classification of SQL-injection attacks and countermeasures". In: *Proceedings of the IEEE international symposium on secure software engineering*. Vol. 1. IEEE, pp. 13–15.

Hindocha, Neal and Eric Chien (2003). "Malicious threats and vulnerabilities in instant messaging". In: *Virus Bulletin Conference, vb2003*.

Holmes, Aaron (2021). *533 million Facebook users phone numbers and personal data have been leaked online*. https://smarturl.it/fb_leak_2021. [Online; accessed 15-Dec-2021].

Ilchenko, V. P. (2017). "Using telegram bots for business". In.

ISO, BSEN and BRITISH STANDARD (2010). "Ergonomics of human-system interaction". In.

Job, J, V Naresh, and K Chandrasekaran (2015). "A modified secure version of the Telegram protocol (MTProto)". In: *2015 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*. IEEE, pp. 1–6.

Johnson, Don, Alfred Menezes, and Scott Vanstone (2001). "The elliptic curve digital signature algorithm (ECDSA)". In: *International journal of information security* 1.1, pp. 36–63.

Jones, M, J Bradley, and N Sakimura (2015). "Rfc 7519: Json web token (jwt)". In: *IETF. May*.

Jones, Michael, Brain Campbell, and Chuck Mortimore (2015). "JSON Web Token (JWT) profile for OAuth 2.0 client authentication and authorization Grants". In: *May-2015.{Online}. Available: https://tools. ietf. org/html/rfc7523*.

Bibliography

Josefsson, Simon (2003). *RFC3548: The Base16, Base32, and Base64 Data Encodings*.

Kappenberger, Reiner (2012). "The true cost of self-signed SSL certificates". In: *Computer Fraud & Security* 2012.9, pp. 14–16.

Lee, Jeeun et al. (2017). "Security analysis of end-to-end encryption in Telegram". In: *Simposio en Criptografía Seguridad Informática, Naha, Japón. Disponible en https://bit.ly/36aX3TK*.

Luo, Xin et al. (2011). "Social engineering: The neglected human factor for information security management". In: *Information Resources Management Journal (IRMJ)* 24.3, pp. 1–8.

Malan, Ruth, Dana Bredemeyer, et al. (2001). "Functional requirements and use cases". In: *Bredemeyer Consulting*.

Mango Messenger Figma, Figma (2021). *Mango Messenger diagrams Figma*. https://www.figma.com/file/cGtLvO1JuJgWmnzaSbM8Re/Mango-Thesis-Screens. [Online; accessed 12-December-2021].

Mannan, Mohammad and Paul C Van Oorschot (2005). "On instant messaging worms, analysis and countermeasures". In: *Proceedings of the 2005 ACM workshop on Rapid malcode*, pp. 2–11.

Ong, Shyue Ping et al. (2015). "The Materials Application Programming Interface (API): A simple, flexible and efficient API for materials data based on REpresentational State Transfer (REST) principles". In: *Computational Materials Science* 97, pp. 209–215.

Proctor, Frederick M (1999). "Linux, Real-Time Linux, & IPC." In: *Dr. Dobb's Journal: Software Tools for the Professional Programmer* 24.11, pp. 32–36.

Rapp, Axel (2021). *Web site security maturity of the European Union and its member states: A survey study on the compliance with best practices of DNSSEC, HSTS, HTTPS, TLS-version, and certificate validation types*.

Rasche, Peter et al. (2016). "Building and Exchanging Competence Interdisciplinarily: Design Patterns as Domain Mediator". In: *Proceedings of the International Symposium on Human Factors and Ergonomics in Health Care*. Vol. 5. 1. SAGE Publications Sage CA: Los Angeles, CA, pp. 19–24.

Rising, Linda (1998). "Design patterns: Elements of reusable architectures". In: *The Patterns Handbook: Techniques, Strategies and Applications*, pp. 9–13.

Rompel, John (1990). "One-way functions are necessary and sufficient for secure signatures". In: *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pp. 387–394.

Bibliography

Schillinger, Fabian and Christian Schindelhauer (2019). "End-to-end encryption schemes for online social networks". In: *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*. Springer, pp. 133–146.

Shay, Richard et al. (2010). "Encountering stronger password requirements: user attitudes and behaviors". In: *Proceedings of the sixth symposium on usable privacy and security*, pp. 1–20.

Simmons, Gustavus J (1979). "Symmetric and asymmetric encryption". In: *ACM Computing Surveys (CSUR)* 11.4, pp. 305–330.

Sušánka, Tomáš and Josef Kokeš (2017). "Security analysis of the Telegram IM". In: *Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium*, pp. 1–8.

Tiwari, Yash and Mallika Tiwari (2015). "A study of SQL of injections techniques and their prevention methods". In: *International Journal of Computer Applications* 114.17.

Wang, Mao-Yin et al. (2004). "An HMAC processor with integrated SHA-1 and MD5 algorithms". In: *ASP-DAC 2004: Asia and South Pacific Design Automation Conference 2004 (IEEE Cat. No. 04EX753)*. IEEE, pp. 456–458.

Wiener, Michael J (1990). "Cryptanalysis of short RSA secret exponents". In: *IEEE Transactions on Information theory* 36.3, pp. 553–558.

Young, Greg (2010). *CQRS and Event Sourcing*. https://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf. [Online; accessed 12-December-2021].

List of functional requirements

Registration user stories

- As an unregistered user, I want to tap “Register” so that I see the registration form and register myself.
- As an unregistered user, I want to use my phone number to register so that my account is tied to my phone number.
- As an unregistered user, I want to use my e-mail address to register so that my account is tied to my e-mail address.
- As an unregistered user, I want to add a display name during registration so that other users can find me using it.
- As an unregistered user, I want to choose how to receive the registration confirmation via SMS or e-mail so that notification is sent to me via SMS or e-email.
- As an unregistered user, I want to receive the registration confirmation via SMS or Email so that I can activate my account.
- As a registered user, I want to confirm my email address so that I get confirmation link via email I provided.
- As a registered user, I want to confirm my phone number so that I use specified form to do it.

Authentication user stories

- As a registered user, I want to authenticate myself using both combinations email-password and phone-password so that I use the specified form with two inputs.
- As a registered user, I want to restore my password if I forget it so that I use specified form and restore my password.
- As an authenticated user, I want my session on each device to least 7 days so that after 7 days of inactivity device will be logged out automatically.

Bibliography

Managing contacts user stories

- As an authorized user, I want to see my contact list so that there is a list of users who are my contacts.
- As an authorized user, I want to search users so that I write user display name or phone number of e-mail address to specified input, click "Search user" button and see results.
- As an authorized user, I want to add other user to my contact list so that I click "Add contact" button on user profile and add him to my contact list.
- As an authorized user, I want user search input to accept empty or whitespace queries so that all users displayed as search result.
- As an authorized user, I want to remove the user from my contact list so that I click "Remove contact" button on user profile and remove him from my contact list.
- As an authorized user, I want to navigate to private chat with the user from my contact list so that I click "Message" button at user profile and get navigated to the private chat with him.
- As an authorized user, I want to tap "Start chat" so that create chat with specified user/contact

Sending messages and media to individuals user stories

- As an authorized user, I want to send a text message so that another user sees my message.
- As an authorized user, I want to add an attachment to the message so that another user sees the message with attachment.
- As an authorized user, I want to add an emoji to the message so that another user sees the message with emoji.
- As an authorized user, I want to tap "Edit" on my message so that message I edited is changed immediately in the chat.
- As an authorized user, I want to tap "Delete" on my message so that message immediately disappears from the chat.

- As an authorized user, I want to share secret messages with users from my contact list so that our are messages encrypted for anyone else including system administrators.
- As an authorized user, I want each new message in private chats I participate to be displayed immediately in real-time so that I do not reload page.

Navigation user stories

- As an authorized user, I want to navigate between the pages so that there is a menu on the UI.

Creating and managing groups user stories

- As a registered user, I want to tap "Create channel" so that I create a new channel of the one of the types: Private channel, Public channel, Readonly channel.
- As a registered user, I want to tap "Start direct chat" so that I create a new direct chat with specified user.
- As a registered user, I want to tap "Start secret chat" so that I create a new secret chat with specified user.
- As a registered user, I want to join public groups so that I click button "Join group" to join the group.
- As a registered user, I want to tap "Archive" so that I archive the specified chat or channel.
- As a registered user, I want to tap "Un-archive" so that I un-archive the specified archived chat or channel.
- As a registered user, I want my secret chats to be device-specific so that I can see a secret chat only on the device that I used to start this chat.
- As registered user, I want to tap "Leave" so that I leave from specified chat or channel.

Sending messages and media to groups user stories

- As an authorized user, I want to send a text message so that other members of the group see the message I sent.

- As an authorized user, I want to add an attachment to the message so that other members of the group see the message with attachment I sent.
- As an authorized user, I want to add an emoji to the message so that other members of the group see the message with emoji I sent.
- As an authorized user, I want to tap "Edit" on my message so that other members of the group see the message I edited.
- As an authorized user, I want to tap "Delete" on my message so that my message is deleted for all members of the group.
- As an authorized user, I want to search public groups by title so that I enter display name to specified field, click button "Search chats" and see results.
- As an authorized user, I want each new message in groups I participate to be displayed immediately in real-time so that I do not reload page.

Viewing messages history user stories

- As an authorized user, I want to view a message history of particular chat or group so that I see a list of my active chats on the UI.
- As an authorized user, I want to search messages in particular chat so that I see the results in messages window of the chat.

Managing profile settings user stories

- As an authorized user, I want to update my personal information in profile settings so that other users my updated personal information.
- As an authorized user, I want to update my social network links in profile settings so that other users my updated social media.
- As an authorized user, I want to change my profile picture so that all other users will see updated one.
- As an authorized user, I want reset password, so that my password will change.
- As an authorized user, I want to tap "Logout" button so that current device will be logged out from the system.

Bibliography

- As an authorized user, I want to tap "Logout all" button so that all my authorized devices will be logged out from the system.
- As an authorized user, I want to navigate to personal information page so that I want see my profile picture.
- As an authorized user, I want to navigate to personal information page so that I want see my personal information.
- As an authorized user, I want to tap "Reset" so that I want reset my updated (not saved) personal information.
- As an authorized user, I want save my updated personal information so that users see it.

List of non-functional requirements

- Graphic user interface of the system should be well organized. To fulfill this requirement, we follow an ISO 9241–161:2010 (en) Ergonomics of human-system interaction standard [ISO and STANDARD, 2010].
- The system should have well performance, which meant to respond it at least 1 second. User should have a device with at least 6 GB RAM and CPU with 1.8 GHZ, 100 Mbps internet connection. Server must have the following hardware: Intel 1+ GHz 2 Cores server processor, 2GB DDR4 memory, NVME or SAS server disk with a minimum capacity of 1.6 GB.
- The unique, unambiguous identifier of users in the system is the username. It is set in the profile settings.
- The Web UI must be well displayed with the following browsers, in the versions current at the date of receipt of the system or, depending on technical possibilities, with the latest versions that support correct operation of the system:
  - Google Chrome 72.0.36.
  - Mozilla Firefox 64.0.2.
  - Microsoft Edge 17.17134.
- The system shall force users to use passwords with a minimum length of 8 characters and using at least one capital letter and one number and one special symbol.
- The Web UI must be compatible to use on mobile device screens with a minimum width of 600 pixels.
- The Web UI must be compatible to use on desktop or laptop device screens with a minimum display width of 1024 pixels.